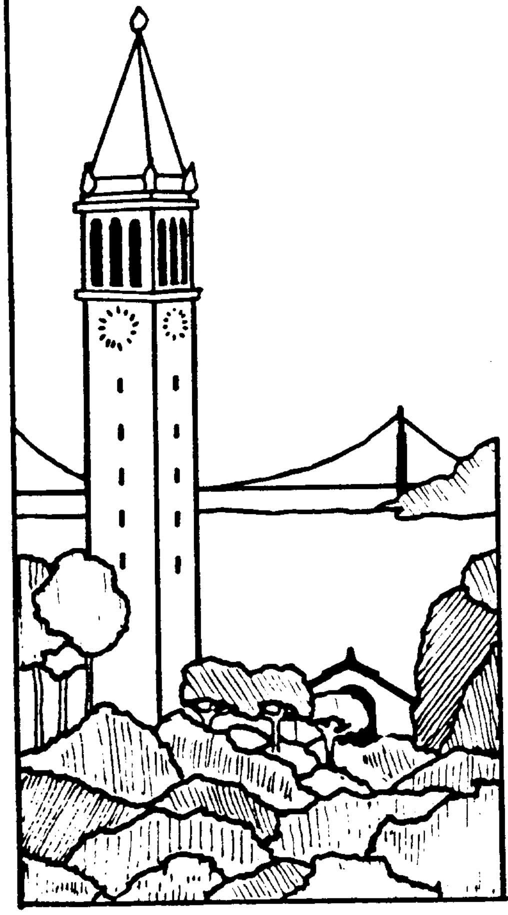
Knowledge-Based Distributed Systems Management

Joseph Pasquale



Report No. UCB/CSD 86/295

June 1986
PROGRES Report No. 86.1

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **FEB 1986**	2. REPORT TYPE	3. DATES COVERED **00-00-1986 to 00-00-1986**
4. TITLE AND SUBTITLE **Knowledge-Based Distributed Systems Management**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Berkeley,Department of Electrical Engineering and Computer Sciences,Berkeley,CA,94720**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Distributed systems characterized by a high degree of inter-computer resource sharing generally perform better if resources are managed utilizing as much knowledge of the current global state of the system as possible. Decentralized resource management schemes have been preferred over centralized schemes for reasons of reliability, autonomy, speed, and symmetry. Yet, distinct computers in a distributed system often view the global system state quite differently. Consequently, decisions which produce system-wide effects made by distinct computers can often conflict, invariably causing inefficiency in resource management and therefore leading to downgraded performance. To address these and related problems, a system is proposed which provides the following: * a mechanism for monitoring events of interest in a distributed system; * a mechanism for distributing monitored data throughout the distributed system; * a mechanism which uses heuristic-based specifications to interpret received monitored data from remote sources so that appropriate actions can be taken when necessary. The novelty and power of the proposed system lies in its application of expert system technology to deal with uncertain, incomplete, erroneous and out-of-date observation data which is inevitable when one tries to efficiently monitor remote events in a distributed system.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**17**	

Knowledge-Based Distributed Systems Management

Joseph Pasquale

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California
Berkeley, CA 94720
February 12, 1986

Abstract

Distributed systems characterized by a high degree of inter-computer resource sharing generally perform better if resources are managed utilizing as much knowledge of the current global state of the system as possible. Decentralized resource management schemes have been preferred over centralized schemes for reasons of reliability, autonomy, speed, and symmetry. Yet, distinct computers in a distributed system often view the global system state quite differently. Consequently, decisions which produce system-wide effects made by distinct computers can often conflict, invariably causing inefficiency in resource management and therefore leading to downgraded performance.

To address these and related problems, a system is proposed which provides the following:

* a mechanism for monitoring events of interest in a distributed system;
* a mechanism for distributing monitored data throughout the distributed system;
* a mechanism which uses heuristic-based specifications to interpret received monitored data from remote sources so that appropriate actions can be taken when necessary.

The novelty and power of the proposed system lies in its application of expert system technology to deal with uncertain, incomplete, erroneous and out-of-date observation data which is inevitable when one tries to efficiently monitor remote events in a distributed system.

This work was supported by an I.B.M. Fellowship and by the Defense Advanced Research Projects Agency (DoD), monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the author and should not be interpreted as representing official policies, either expressed or implied, of I.B.M., the Defense Research Projects Agency or of the US Goverment.

1. Introduction: The General Problem

Remote resource-sharing in distributed systems has become increasingly popular. This development is quite analogous to the situation experienced twenty years ago, when time-sharing was introduced. Time-sharing allowed expensive and scarce resources (eg. CPU, primary memory, disks, tapes) to be shared in an orderly manner among many user processes. Networks connecting computers (along with appropriate protocols) offer user processes the capability of accessing resources which are remote, or located on different computing nodes of the network.

There is a tremendous difference, though, in managing resource-sharing on the same computer and in managing resource-sharing on remote computers, particularly when one considers where the locus of control is in both situations. A computer's operating system is (for the most part) the single locus of control or the single source of decision-making for that computer's resources. Many distributed systems exist where decision-makers mutually manage all the resources, the decision-makers being the nodes participating in the resource-sharing. This decentralized scheme of control is hailed for such reasons as reliability (no single point of failure), autonomy (no one should have to take orders from anyone else), speed (many can do the job faster than one), and symmetry (adding or deleting nodes is easy and natural). Yet, just as in social systems, having many decision-makers work concurrently on the same problem can lead to many conflicting decisions, especially when each decision-maker sees the problem situation differently. Prescribing that decision-makers somehow agree to produce unified decisions (e.g., voting) is normally not possible due to time and communication constraints. Consequently, in retrospect, it may seem easier and more effective simply to elect a leader to make all the decisions, and give up whatever benefits decentralized control can offer.

I am proposing to give decentralized control another chance by trying to focus on why conflicting decisions occur and how to minimize their frequency. Assuming that conflicting decisions are due to inconsistent (or non-existent!) views of the global state of the distributed system, the method of attack is to make relevant portions of the global state more accessible to the decision-making nodes, and to make the nodes more intelligent.

2. Design of a Knowledge-Based Distributed Systems Manager

A new scheme is proposed for managing resources in distributed systems. This design is general in that it can be applied to many distributed resource-sharing problems such as load-balancing, message-routing, and distributed file placement. Of course, the efficiency of the implementation of this design will depend on the particular application and the architecture of the distributed system. Consequently, any implementation issues will focus on the application under investigation.

3. The Structure of a Knowledge-Based Distributed Systems Manager

The Knowledge-based Distributed Systems Manager has three subsystems:

(1) Local State Monitors;

(2) State Distributors;

(3) Expert Managers.

Briefly, Local State Monitors provide a mechanism for monitoring events of interest in the distributed system. Each computing node has a Local State Monitor which monitors state variables inside that node. To communicate state information to other nodes, State Distributors are provided. Thus, each computing node has a State Distributor which sends local state information to other nodes and also acquires state information about remote nodes. Finally, the mechanism which interprets all this local and remote state information is the

Expert Manager. Expert Managers are small expert systems, and many can reside on a single node. Each Expert Manager knows how to deal with one application very well. For example, there may be an Expert Manager on every node for message routing; there may be another Expert Manager for balancing the load on a set of computers.

A more detailed discussion of the above mechanisms follows.

4. Local State Monitors

At the lowest level of this system resides a mechanism for monitoring those events of interest which take place inside a node. Monitors can be of the hardware or software variety. I am assuming that software monitors will be there since they are relatively quick and easy to implement. Typically, events are not directly observable by software monitors, but the side effects they produce are directly observable. For example, an event of interest might be the arrival of a new process which joins the CPU queue. This event is not directly observable (if the CPU is being used to observe an event, it cannot possibly be queueing a process, assuming software monitors are not allocated dedicated processors), but if a side effect of queueing a process is to increment a counter, the counter's value can certainly be observed to have changed.

Consequently, the technique suggested to monitor events is to sample variables which get affected in some way (e.g., incremented) when the event of interest occurs. This technique has the main advantage that the monitor can be either a distinct process or a distinct part of the operating system kernel which simply reads special variables in the kernel. Regardless of whether it is a process or a part of the kernel, the monitor is a self-contained piece of software which can be easily inserted or deleted, and is simple to understand, modify, and debug. Of course, its disadvantage is that, if we do not sample values at a sufficiently high rate, the monitor may not detect changes which are occurring very rapidly. This turns out to be a minor problem because events of interest to remote nodes will not change that rapidly. The speed of response to those changes is limited by the communication time between nodes, which is sufficiently large relative to the time between samples. For example, if it takes on the order of 1/10 second to send, receive, and process a message that contains sampled data, it makes no sense to sample at a rate of, say, one observation per millisecond.

A *State Variable* is a variable kept in a computer's primary memory representing a piece of state information about some object within that computer. Examples of state variables are: the length of the CPU ready-process queue; the length of network input and output queues; the amount of free vs. used primary memory; the amount of free space in a local file system.

State variables are typically defined during the implementation of an operating system, but can also easily be added at a later time. These state variables may correspond to hardware sensors, software probes, or statistics which are functions of sensor and probe samples. The main point is that they represent state information about some function in the machine where they reside.

A *Local State Monitor* resides inside each computer node and monitors state variables which will eventually be communicated to other nodes in the distributed system. The *Local State* of a computer node is the collection of all state variables currently being monitored. In particular, the Local State Monitor provides the following:

* the sampling of designated state variables and an interface for obtaining state variable sample statistics;
* a user interface which allows specification of what state variables are to be monitored and what their system-wide interpretation is to be;

u

* a tracing capability which records a history of state variable samples in a file.

Thus, the Local State Monitor is a self-contained unit which will provide its services to other subsystems through a specific interface.

As an example, a computer node's Local State may consist of the following three state variables:

(1) length of network input queue;

(2) length of network output queue;

(3) length of ready-process queue.

In general, these state variables would actually be time-smoothed versions of the instantaneous corresponding queue lengths. Even this small Local State would be useful for simple centralized or decentralized dynamic routing algorithms.

The Local State Monitor maintains a database of statistics about state variables currently being monitored. These statistics can be used to infer properties about the past behavior of each state variable. Such information may then be used for predictive purposes. For example, the following pieces of statistics might be kept about each state variable:

* the current sampled value;
* the geometric average value (e.g., x[n] + x[n-1]/2 + x[n-2]/4 + ...);
* the change between the current and the previous samples (e.g., c[n] = x[n] - x[n-1]);
* the geometric average change (e.g., c[n] + c[n-1]/2 + c[n-2]/4 + ...);
* a time-stamp of when the sample was updated.

This set of statistics provides information about the state variable's instantaneous value, its typical value and variability as a function of time, and how stale the information is. One can efficiently implement a generator of these statistics using 2 shift and 3 add operations, along with the required memory reads and writes. Thus, even if there were 100 state variables being monitored every 1/10 second and the time to perform a basic instruction were 1 microsecond, the percentage of CPU time used to perform this monitoring would be on the order of a tenth of a percent.

The Local State Monitor also manages the database of state-variable samples received from remote nodes through the *State Distributor*, to be described in the next section. Consequently, information about remote state variables can also be accessed from the Local State Monitor through the same interface used to access local state variable information. In this case, an identifier of the remote node would be passed along with the other interface parameters.

Descriptions of other systems which have used similar ideas can be found in [Borysowich82] [Lewis80] [Tarnay80] [Terplan81] [Terplan83]. A system which does a similar monitoring function at the user level is Metric [McDaniel75]. In fact, the way Metric allows the type definition of events could be adopted for the definition of state variables. As in Metric, a particular type of state variable would be interpreted the same way throughout the distributed system once conventions are developed.

5. State Distributors

Once state variables of interest are sampled, there must be a mechanism by which they can be communicated to remote nodes. This mechanism must be highly efficient since the data being transmitted becomes less useful as time passes on. Although this depends very much on the particular structure of the distributed system and on the available hardware, the design must be able to take advantage of any implementation-specific efficiencies that might be achieved.

A *Relevant Local State* is the collection of state variables in a node which are of interest to remote nodes. A *Relevant Local State Snapshot* is the collection of the most current sampled values of state variables in the Relevant Local State. A *State Distributor* is the mechanism which distributes Relevant Local State Snapshots to other nodes in the distributed system. There is a State Distributor in all nodes (participating in resource sharing), and State Distributors communicate two types of information between each other:

(1) the Relevant Local State Snapshots of the node they reside in;

(2) specifications of what state variables in remote nodes are of interest (i.e., what state variables should be included in a remote node's Relevant Local State).

When a State Distributor *receives* a Relevant Local State Snapshot from a remote node, it communicates this information to the Local State Monitor, which provides the interface to other subsystems for accessing the data. The collection of all Relevant Local State Snapshots which a node collects from other nodes (along with its own Local State) is called the node's *Relevant Global State*. Thus, when a request is made to the Local State Monitor for the value of a particular state variable, one may also specify the node of interest. For example, one may request the value of the local CPU queue length, or the value of a remote node's CPU queue length if the node is specified.

A State Distributor makes use of the interface provided by the Local State Monitor residing on the same computing node. This is how it obtains those values of the Local State which become the Relevant Local State Snapshot to be distributed.

Consequently, it is the function of all the State Distributors in the distributed system cooperatively to produce a kind of "raw" global system state for each node, namely the Relevant Global State. Note that this raw global system state is very approximate; in fact, each node could easily have a different version of this state. Thus, this raw global system state must go through a level of interpretation or filtering so that a better approximation of the true global state is available. This will be the subject of the next section.

As mentioned above, the implementation of State Distributors is dependent on the architecture of the distributed system and on the types of distributed computations which will make use of Relevant Global States. For example, a routing algorithm for a distributed system which consists of an inter-network (i.e., a network of networks) of computing nodes may require *all* nodes in the inter-network to know state information about each other. Other algorithms require knowledge of state information only about nodes in the same network, with gateways taking care of inter-network traffic, and still other algorithms require only that neighboring nodes exchange state information. Consequently, a State Distributor for such applications must potentially be able to distribute Relevant Local State Snapshots to a large number of nodes, many of which can be very distant from each other.

Another application might be load balancing, where the combined load of the computing nodes on a single local-area network is to be balanced across the nodes. In this case, the State Distributors may take advantage of the broadcast capability of the local-area network to transmit load information. Further optimizations are possible if smart network interfaces which have DMA capability are used; broadcasting incurs very little overhead in this case. In yet another scenario, the State Distributor may make use of a distributed file system if broadcasting is not possible or undesirable.

The idea of periodically sending performance data between nodes is well-established in the literature, especially for network monitoring systems [Abrams78] [Alton81a] [Alton81b] [Barchanski81] [Bartz83] [Bernstein83a] [Bernstein83b] [Buck78] [Caneshi81] [Grange77] [Herman82] [McCrea82] [Murphy81] [Murphy82] [Terplan82]. Yet, there are many unresolved issues such as the frequency of sending, whether broadcasting or multicasting should be used, and host interference and network interference due to the distribution process.

6. Expert Managers

Local State Monitors and State Distributors implement the mechanisms for exchanging observations about events occurring in all the nodes of a distributed system. The amount of information present in the raw observation data which a node receives can be overwhelming, to the point of being almost useless. For example, if there are 100 nodes and 10 relevant state variables are sampled at each node, and each state variable has 100 different values (not a very complex system), we would face a state space of size 100,000! Specifying what to do for each case would be an exhausting process, and the prospect of adding new relevant state variables would be disastrous.

Consequently, what is needed is a mechanism which will reduce this unwieldy state space to a small set of meaningful states. This mechanism must also be able to deal with the fact that much of the observed data is imperfect; not only can the data be noisy, but it will have aged since the time it was generated (by sampling a memory location, a process that introduces further uncertainty) and then communicated. This uncertainty must at least be expressible so that decisions made using the imperfect data take uncertainty into account.

An *Expert Manager* is a mechanism which addresses these problems through the use of techniques borrowed from expert system technology. In fact, an Expert Manager is really a small expert system. Its function is to analyze the current Relevant Global State (all the most recent data observed from all the nodes in the distributed system) and then make reasonable inferences about what is happening in the distributed system. Upon concluding that certain actions need to be taken, it can directly activate processes specifically designed to carry out these actions. Before continuing, a discussion about expert systems is in order.

6.1. Expert Systems

A key characteristic of an expert system is its use of *heuristics* to search for solutions to problems. These heuristics are basically "rules-of-thumb" which experts in the problem area go by when they solve problems. Moreover, the specification language of an expert system is typically a set of heuristics in the form of IF-THEN rules. These expert system languages are declarative (i.e., they specify *what* to do, as opposed to procedural languages, which specify *how* to do something). Most common programming languages are procedural. It is the role of Expert Managers to be *policy* makers. Expert Managers indicate *what* nodes should be doing cooperatively to solve a distributed problem, and not *how* the underlying *mechanisms* carry the policies out. Consequently, it is natural that a declarative rule-based language be used to program Expert Managers.

Expert systems are composed of a *Knowledge Base*, in the form of heuristics or a set of rules, and an *Inference Engine*, a mechanism which makes inferences through the application of the knowledge-base rules. An expert system operates by accepting inputs, and then applies appropriate rules to make conclusions about the inputs. Rules are then applied to the conclusions to make further conclusions, this process continuing until the problem is solved. Expert systems have other interesting and useful features, which make them ideally suited to implement Expert Managers. These other features found in various expert systems are:

* Forward-Chaining Control
* Action Activation
* Goal-Oriented Operation
* Ability to Deal with Uncertainty

6.2. An Example

To explain how an Expert Manager would work, consider the following example of a load balancing Expert Manager. First, a brief description of distributed system state monitoring and distribution is given, followed by how Expert Managers make use of the system state.

6.3. State Monitoring and Distribution

A local-area network has many computing nodes and it is desired that the load be roughly balanced across all nodes, the idea being that it would be poor management to have one computing node overworked if there were other nodes which were completely idle. Assume that a reasonable metric for the load of a computer is the length of the CPU ready-process queue. Consequently, we define a state variable, called LA for Load Average, which is a statistic constructed by time-averaging samples of the instantaneous CPU ready-process queue length. This state variable is monitored by the Local State Monitor inside each computer node, and its values are made available to other nodes through the State Distributor.

6.4. Rules and State Interpretation

An Expert Manager dedicated to load balancing could now interpret the LA state variable in the following way:

(1) IF LA(X) > 8 THEN State(X) is OVERLOADED

(2) IF 2 < LA(X) < 8 THEN State(X) is BUSY

(3) IF LA(X) < 2 THEN State(X) is IDLE

These are typical rules which might be programmed into an Expert Manager. Rule 1 translates to: if the load average of node X is greater than 8, consider the state of node X to be OVERLOADED. Substituted for X are all nodes which communicate their Relevant States to the node on which the Expert Manager resides.

Here, nodes are considered to be in one of three states. Thus, a simple function that can be carried out by the Expert Manager consists of taking a large state space defined by all the local and remote state variables received, and conglomerate them into a small set of meaningful states (in this case, three) through simple rules.

IF-THEN rules are the most common way of expressing knowledge in expert systems. One of the first expert systems to express rules in this fashion was the MYCIN system [Shortliffe76]. Other notable examples are PROSPECTOR [Duda78], PUFF [Freiherr80], and ROSIE [Fain82].

6.5. Inferring Causal Relationships

The Expert Manager, given the above three rules, can then construct the following rules on its own:

(4) IF LA(X) is increasing AND State(X) is IDLE THEN State(X) will become BUSY

(5) IF LA(X) is increasing AND State(X) is BUSY THEN State(X) will become OVERLOADED

(6) IF LA(X) is decreasing AND State(X) is OVERLOADED THEN State(X) will become BUSY

(7) IF LA(X) is decreasing AND State(X) is BUSY THEN State(X) will become IDLE

This was accomplished by a *meta-rule* of the Expert Manager, which notices that the range of LA is divided into intervals, and each interval is a state. It then constructs rules 4 through 7.

The idea of a meta-rule, a rule about other rules, is evident in many expert systems such as MYCIN [Shortliffe76], DENDRAL and META-DENDRAL [Buchanan78] [Lindsay80], and HEARSAY-III [Balzar80], and much of their power stems from the use of meta-rules. Rules 4 through 7 are actually a very simple form of the application of meta-rules. Here, a meta-rule exists specifically for the case when states are assigned to intervals of some state variable.

6.6. Forward Chaining and Actions

Rules can also express that actions should take place. For example:

(8) IF State(ME) is OVERLOADED AND State(X) is IDLE THEN ACTIVATE (OFFLOAD (X))

This rule simply states that if the state of the local node is OVERLOADED and the state of some other node is IDLE, then a process called OFFLOAD should be activated. This process is not really part of the Expert Manager. It is a process independently implemented, whose function is to offload processes located on the same node on which it resides to a node provided as a parameter. Thus, the Expert Manager makes the *policy decision* to offload, and an independent process implements the *mechanism*.

As the Expert Manager executes, it continuously tries to match the antecedents of all its rules. If the local LA happened to be 10, and the LA on node A was 1, the Expert Manager would conclude that State(ME) is OVERLOADED and State(A) is IDLE. This would in turn fire rule 8, which causes the offloading process to take corrective action. The control mechanism to accomplish this is commonly called *Forward Chaining*.

6.7. Backward Chaining, and Goals

An Expert Manager may also work in a goal-oriented mode. Consider the following additional rules:

(9) IF LA(ME) - LA(X) > 3 THEN SystemState is UNBALANCED

(10) IF LA(ME) - LA(X) < 3 THEN SystemState is BALANCED

One should immediately notice that inferences about causality can be made here. The following additional rules are generated:

(11) IF (LA(ME) - LA(X)) is increasing AND SystemState is BALANCED THEN SystemState will become UNBALANCED

(12) IF (LA(ME) - LA(X)) is decreasing AND SystemState is UNBALANCED THEN SystemState will become BALANCED

The Expert Manager may now be placed in a mode to achieve a goal by installing the following directive:

(13) Make SystemState BALANCED

If SystemState's value is BALANCED, the Expert Manager does nothing. If SystemState's value is or becomes UNBALANCED, the Expert Manager tries to determine how to make it BALANCED. To do this, the Expert Manager uses backward chaining control to accomplish the above goal. This is done by looking at the THEN parts of each rule for "SystemState will become BALANCED". Observe that rule 12 applies. The Expert Manager tries to make the antecedent of rule 12 true, and this is how a goal is accomplished. Since SystemState is currently UNBALANCED, one part of the antecedent is true. If LA(ME) - LA(X) is in fact decreasing for some node X, it will believe that eventually, the SystemState will become BALANCED, and the goal is achieved. If LA(ME) - LA(X) is not decreasing for any node X, it will inquire (a system operator or administrator):

"How can I make 'LA(ME) - LA(X) is decreasing' TRUE?"

A new rule may then be added:

(14) IF ACTIVATE (OFFLOAD (X)) THEN LA(ME) - LA(X) is decreasing

Using 14, the Expert System activates the OFFLOAD process and is then satisfied that the goal will eventually be accomplished.

This backward chaining type of control is exactly how the MYCIN expert system works [Shortliffe76].

6.8. Dealing with Uncertainty

Perhaps the greatest benefit that can be derived by using expert system technology to solve distributed systems management problems is to incorporate techniques of dealing with uncertainty. Briefly, the idea is that every rule may be given a certainty factor. This factor indicates how sure we are that this rule truly works. Certainty factors usually range from -1 to 1, where -1 means we are sure the rule is false, 1 means we are sure it is true, and 0 means total uncertainty. Thus, we have a way of expressing how certain the knowledge is in the knowledge-base.

Not only should we be able to express that rules are uncertain, we need especially to express that values of monitored state variables from remote nodes are uncertain. This uncertainty can vary in time: for example, an uncertainty function for a state variable can depend on its age and variability. As the data gets older or staler, we become less confident that it represents the true value at the remote node. It is also natural to include variability in the uncertainty factor; the more variable the data is, the less certain we are it is valid as time passes.

7. Summary

A proposal for a Knowledge-Based Distributed Systems Manager has been described. It is composed of three subsystems: Local State Monitors, State Distributors, and Expert Managers, which respectively monitor, distribute, and interpret the global state of the distributed system. The novelty and power of this system lies in the way Expert Managers make use of the information provided by Local State Monitors and State Distributors. Techniques from expert systems are incorporated in Expert Managers, namely state interpretation, forward chaining, goal-oriented operation, and dealing with uncertainty using certainty factors, which allow intelligent interpretation of raw global state information. These techniques enable us naturally to express the types of functions needed for arriving at policy decisions when managing resources in a distributed system where the state of remote resources is not known with complete certainty.

8. Acknowledgements

I wish to express deep gratitude to my research advisor, Professor Domenico Ferrari, who continually gave me guidance and advice. This research exists only because of his constant encouragement. I am also indepted to Dr. Luis Felipe Cabrera, who during the early stages of this research inspired many key ideas.

9. Bibliography

[Abrams78]

Abrams, M.D., and Hayden, H.P., "Application of a Network Monitor to the Selection of a Time Shared Computing System," *14th meeting of the Computer Performance Evaluation Users Group (Boston, MA)*, pp. 15-25, Nat. Bur. Standards, Washington D.C., USA, Oct. 1978.

[Alton81a]

Alton, B., Baker, S., Horn, C., Kenyon, S., Patel, A., Purser, M., and Sheehan, J., "Networking Experience at Trinity College, Dublin," *Networks from the User's Point of View. Proceedings of the IFIP TC-6 Working Conference COMNET '81 (Budapest, Hungary)*, pp. 205-15, North-Holland, Amsterdam, Netherlands, May 1981.

[Alton81b]

Alton, B., Patel, A., Purser, M., and Sheehan, J., "The Performance of a Packet Switched Network - A Study of EURONET," *Performance of Data Communication Systems and their Applications. Proceedings of the International Conference (IFIP, IEEE, Paris, France)*, pp. 379-89, North-Holland, Amsterdam, Netherlands, Sept. 1981.

[Armstrong78]

Armstrong, T.R., "Minicomputer Network Monitoring Lessens Fear of 'Dependency'," *Data Management (USA)*, pp. 33-5, Dec. 1978.

[Barchanski80]

Barchanski, J.A., and Muehleisen, T., "Quality of Service Monitoring of the Open Systems Architecture Transport Layer," *Tanulmanyok Magy. Tud. Akad. Szamitastech. and Autom. Kut. Intez. (Hungary)*, pp. 33-59, 1980.

[Barchanski81]

Barchanski, J.A., and Muehleisen, T., "Basic Concepts of the Quality of Transport Service Monitoring in an Inter-University Computer Network," *Performance of Data Communication Systems and their Applications. Proceedings of the International Conference (Paris, France)*, pp. 241-50, North-Holland, Amsterdam, Netherlands, Sept. 1981.

[Barnes82]

Barnes, A.C., and Graves, J., "Operational and Maintenance Experience of PSS," *Pathways to the Information Society. Proceedings of the 6th International Conference on Computer Communication (London, England)*, pp. 237-40, North-Holland, Amsterdam, Netherlands, Sept. 1982.

[Bartz83]

Bartz, W.S., and Patterson, R.W., "Total Network Data System: National Network Management," *Bell Syst. Tech. J. (USA)*, pp. 2261-80, Bell Labs., Murray Hill, NJ, Sept. 1983.

[Bernstein83b]

Bernstein, M., Sunshine, C., and Kaufman, D., "A Network Control Center for Broadband Local Area Networks," *Local Networks. Distributed Office and Factory Systems. Proceedings of Localnet '83*, pp. 425-34, Online Publications, Pinner, England, June 1983.

[Bernstein83a]
Bernstein, S.L., and Herman, J.G., "NU: A Network Monitoring, Control, and Management System," *IEEE International Conference on Communications: Integrating Communication for World Progress (ICC '83) (Boston, MA)*, pp. 578-83, IEEE, New York, NY, June 1983.

[Borysowich82]
Borysowich, L.H., and Houghton, B.G., "Network Monitor," *IBM Tech. Disclosure Bull. (USA)*, pp. 543-7, July 1982.

[Buchanan78]
Buchanan, B.G., and Feigenbaum, E.A., "DENDRAL and Meta-DENDRAL: Their applications dimension," *Artificial Intelligence*, pp. 5-24, 1978.

[Buck78]
Buck, D.L., and Hrynyk, D.M., "Software Architecture for a Computer Network Monitoring System," *Performance of Computer Installations. Evaluation and Management (Gardone Riviera, Italy)*, pp. 269-87, North-Holland, Amsterdam, Netherlands, June 1978.

[Caneschi80]
Caneschi, F., Ceccarelli, A., Ferrini, R., Lenzini, L., and Menchi, C., "RPCNET Measurement System: First Experiences," *Atti del Congresso Annuale AICA '80 (Annual Conference AICA '80)*, pp. 29-31, Tecnoprint, Bologna, Italy, Oct. 1980.

[Caneschi81]
Caneschi, F., Lenzini, L., and Manchi, C., "The Behaviour of a Packet Switching Network Running under a Time Sharing Operating System," *Networks from the User's Point of View. Proceedings of the IFIP TC-6 Working Conference COMNET '81 (Budapest, Hungary)*, pp. 397-410, North-Holland, Amsterdam, Netherlands, May 1981.

[Chu83]
Chu, Van, "Monitoring Network Performance," *Computerworld (USA)*, pp. 91-4, May 18, 1983.

[Duda78]
Duda, R.O., Hart, P.E., Barrett, P., Gaschnig, J., Konolige, K., Reboh, R., and Slocum, J., "Development of the PROSPECTOR Consultation System for Mineral Exploration," *Final Rept. SRI Projects 5821 and 6415*, Artificial Intelligence Center, SRI International, Menlo Park, CA, 1978.

[Fain82]
Fain, J., Hayes-Roth, F., Sowizral, H., and Waterman, D., "Programming in ROSIE: An introduction by means of examples," *Tech. Rept. N-1646-ARPA*, Rand Corporation, Santa Monica, CA, 1982.

[Freiherr80]
Freiherr, G., "The Seeds of Artificial Intelligence," *NIH no. 80-2071*, SUMEX-AIM, Washingtion, D.C., 1980.

[Gallo81]
Gallo, A., and Wilder, R.P., "Performance Measurement of Data Communications

Systems with Emphasis on Open System Interconnections (OSI)," *8th Annual Symposium on Computer Architecture (Minneapolis, MN, USA)*, pp. 149-61, IEEE, New York, USA, May 1981.

[Goodier79]
Goodier, M., "Central Management for Distributed Intelligence," *Data Processing (GB)*, pp. 38-9, March 1979.

[Grange77]
Grange, J.L., "Operating the Cigale Packet Switching Network: Concepts, Techniques and Results," *Eurocon '77 Proceedings on Communications (IEEE)*, pp. 44-9, IEEE, New York, NY, May 1977.

[Herman82]
Herman, J.G., and Bernstein, S.L., "Monitoring, Control, and Management of the Defense Data Network," *Conference Record of EASCON 82. 15th Annual Electronics and Aerospace Systems Conference (Washington DC)*, pp. 95-102, IEEE, New York, NY, Sept. 1982.

[Hinden83]
Hinden, R.A., "A Host Monitoring Protocol," *RFC-869*, Bolt Beranek and Newman Inc., Cambridge, MA, Dec. 1983.

[Johnston83]
Johnston, K., "Network Planning," *Syst. Int. (GB)*, pp. 41-2, Oct. 1983.

[Juttelstad83]
Juttelstad, D.P., and Paesano, S., "A Hardware Test-bed for the Evaluation of Distributed Processing Concepts," *Proceedings of the 16th Hawaii International Conference on System Sciences*, pp. 367-76, Hawaii Int. Conference Syst. Sci., Honolulu, HI, Jan. 1983.

[Lewis80]
Lewis, W.E., "Mechanism for Synchronizing Hardware and Software System Performance Monitors," *IBM Tech. Disclosure Bull. (USA)*, pp. 54-5, June 1980.

[Lindsay80]
Lindsay, R.K., Buchanan, B.G., Feigenbaum, E.A., and Lederberg, J., *Applications of Artificial Intelligence for Organic Chemistry: The DENDRAL Project*, McGraw-Hill, New York, NY, 1980.

[Manning81]
Manning, E.G., Wong, J.W., Powell, P.A.D., Radia, S.R., and Tokuda, H., "Shoshin - A Testbed for Distributed Software," *IEEE 1981 International Conference on Communications*, pp. 25.5/1-5, IEEE, New York, USA, June 1981.

[McCrea82]
McCrea, R.C., and Akers, F.A., "Data-Communication Networking: Components, Structure and Performance Analysis," *Computer Performance (GB)*, pp. 126-44, Sept. 1982.

[McDaniel75]
G. McDaniel, "METRIC: a kernel instrumentation system for distributed

environments," *Proc. of the Sixth Symp. on Operating Sys. Principles*, pp. 93-99, Purdue University, November 1975.

[Metcalfe76]
R. M. Metcalfe, and D. R. Boggs, "Ethernet: distributed packet switching for local computer networks," *Comm. of the ACM*, pp. 395-404, July 1976.

[Morency81]
Morency, J.P., "Managing Computer Networks," *Telecommunications (USA)*, pp. 33-4, May 1981.

[Morgan83]
Morgan, A.H., "Overlord: The Changing Role in Switching," *Telecommunications (USA)*, pp. 113-16, Oct. 1983.

[Murphy81]
Murphy, J.L., "Centralized Control and Monitoring of a Distributed Local Network," *IEEE 6th Conference on Local Computer Networks*, pp. 93-9, IEEE, New York, NY, USA, Oct. 1981.

[Murphy82]
Murphy, J.L., and Midkiff, E.L., "Extension of Network Monitoring to a System-Wide Resource Management Function," *Proceedings of Computer Networks Compcon 82, 25th IEEE Computer Society International Conference (Washington D.C.)*, pp. 720-4, IEEE, New York, NY, Sept. 1982.

[Postel82]
Postel, J., Sunshine, C., and Cohen, D., "Recent Developments in the DARPA Internet Program," *Pathways to the Information Society, Proceedings of the 6th International Conf. on Computer Communications*, pp. 975-80, London, Sept. 1982.

[Saito81]
Saito, T., Kato, T., Yano, H., and Inose, H., "Protocol Product Validity by Means of Link Monitoring (Computer Networks)," *IEEE 1981 National Telecommunications Conference. Innovative Telecommunications - Key to the Future (New Orleans, LA)*, pp. F6.1/1-4, IEEE, New York, NY, Nov. 1981.

[Seid83]
Seid, H.A., "The Ins and Outs of Managing a Packet Network," *Data Commun. (USA)*, pp. 149-61, Oct. 1983.

[Shoch79]
Shoch, J.F., and Hupp, J.A., "Measured Performance of an Ethernet Local Network," *Local Area Communications Network Symposium*, May 1979.

[Shortliffe76]
Shortliffe, E.H., *Computer-based Medical Consultation: MYCIN*, American Elsevier, New York, NY, 1976.

[Tan82]
Tan, W., Meng, P., and Hadad, A., "Adaptive Protocol for Worsening Communication Line Conditions," *MILCOM '82. 1982 IEEE Military Communications*

Conference. Progress in Spread Spectrum Communications (Boston, MA), pp. 16.3/1-5, IEEE, New York, NY, Oct. 1982.

[Tarnay80]
Tarnay, K., "Measurement of Computer Networks," *Hungarian Acad. Sci., Budapest*, pp. 10, 1980.

[Terplan79]
Terplan, K., "Performance Measurement of Computer Networks (in GERMAN)," *Angew. Inf. (GERMANY)*, pp. 329-33, Aug. 1979.

[Terplan81]
Terplan, K., "Network Monitor Survey," *Comput. Performance (GB)*, pp. 158-73, Dec. 1981.

[Terplan82]
Terplan, K., "Organizing a Network Administration Center," *EDP Performance Rev. (USA)*, pp. 1-8, Aug. 1982.

[Terplan83]
Terplan, K., "Communications Systems Performance Management," *Comput. Performance (GB)*, pp. 29-34, Mar. 1983.